Submit your work through codeboard:
Cohort A: https://codeboard.io/projects/389732

Task 1 (Guided):
Write a class of objects named **Tube** that remembers information about a circle. It may help you to know that there is a constant named Math.PI storing the value of π, roughly 3.14159.

Class Description
- Constructor: Tube(r, t), to construct a new Tube with given radius and height
- baseCircumference(), returns the distance around the base of the tube
- baseArea(), returns the area occupied by the base of the tube
- area(), returns the area occupied by the tube, you can use previously defined method
- volume(), returns the volume of the tube, you can use previously defined method

In Main class, create an instance of a tube with a radius of 5 units and a height of 10 units. Calculate and print the following information about the tube:
- The circumference of the base.
- The area of the base.
- The total surface area of the tube (including the base).
- The volume of the tube.

Task 2 (Semi-Guided):
You are tasked with implementing a class called "OpoAccount" that models a user account for a mobile payment application called OpoPay. The OpoAccount class will have various properties and methods to manage the account's balance, transactions, and points.

Write a Java class called "OpoAccount" that satisfies the following requirements:
- Upon initialization, the class constructor takes three parameters: phoneNumber (a String), name (a String), and balance (a double). These parameters are used to set the corresponding properties of the account.

- The class provides several methods to interact with the account. The "getBalance()" method returns the current balance of the account. Similarly, the "getPhoneNumber()" and "getName()" methods retrieve the associated phone number and account holder's name, respectively.

- To keep track of the transaction history, the "getTransactions()" method returns a string that contains a

record of all past transactions for the account. This transaction history is updated when certain actions are performed on the account.

- The "topUp" method allows the user to add funds to their account. The specified amount is added to the account's balance, and if the amount is equal to or greater than $100, the account holder earns 10 points for each $100 increment. The transaction is recorded in the transaction history.

- Withdrawing funds from the account is possible using the "withdraw(double amount)" method. If the account balance is sufficient or negative balance is allowed, the specified amount is subtracted from the balance. The transaction is recorded in the transaction history.

- Transferring funds from the current account to another account is facilitated by the "transfer(OpoAccount account, double amount)" method. A transfer fee of $5 is deducted from the current account, and the specified amount is added to the recipient account's balance. Both accounts update their transaction histories accordingly. Additionally, if the transferred amount is equal to or greater than $100, the account holder earns 10 points for each $100 increment.

- If this account object does not have enough money to make the full transfer, transfer whatever money is left after the $5 fee is deducted. If this account has under $5 or the amount is 0 or less, no transfer should occur and neither account's state should be modified.

After OpoAccount class has been properly implemented, create two instances of OpoAccount with different names, phone number and initial balances in Main class. Perform the following operations:

- Top up the first account with $200.
- Transfer $150 from the first account to the second account.
- Withdraw $100 from the second account.
- Print the account details for both accounts, including the account owner's name, balance, and transaction history.

Example output:

```
Account details for User A:
Balance: $145.0
Transaction History: top up of $800.0
transfer of $150.0 to 987654321


Account details for User B:
Balance: $100.0
Transaction History: transfer of $150.0 from 12345678
withdrawal of $100.0
```

Task 3 (Unguided):

You are tasked with creating a text-based monster battle game using Java. The game will involve a player and a monster fighting against each other. Your goal is to implement the game logic and allow the user to control the player's actions during the battle.

Write a Java program that accomplishes the following:

1. Prompt the user to input the player's name and the monster's name.
2. Create a "Human" object representing the player and a "Monster" object representing the monster, using the provided names.
3. Display the initial status of the player and the monster, showing their current hit points and other relevant information.
4. Start the battle with alternating turns between the player and the monster.
5. During the player's turn:
    - Prompt the user to input an action (represented by an integer).
    - If the chosen action is to attack (action 1), call the player's "attack" method and subtract the damage dealt from the monster's hit points.
6. During the monster's turn:
    - Prompt the monster to choose an action (represented by an integer).
    - If the chosen action is to attack (action 1), call the monster's "attack" method and subtract the damage dealt from the player's hit points.
7. Continue the battle until either the player's or the monster's hit points reach zero.
8. If the monster's hit points reach zero first, display a victory message for the player and call the player's "levelup" method.
9. If the player's hit points reach zero first, display a defeat message for the player.

Ensure that your program adheres to the following specifications:

The "Human" class should have the following methods:

- Constructor: Takes the player's name as a parameter and initializes the hit points and other necessary attributes.
    - Hitpoint = 100
    - baseAttack = 10
    - Level = 1
- "attack" method: Calculates and returns the damage dealt by the player's attack(total damage = attack * level).
- "attacked" method: Takes the damage dealt by the monster and subtracts it from the player's hit points.
- "checkStatus" method: Displays the current hit points and other relevant information for the player.

- "levelup" method: Increases the player's level after defeating a monster.
- "dead" method: Displays a message indicating that the player has been defeated.

The "Monster" class should have the following methods
- Constructor: Takes the monster's name as a parameter and initializes the hit points and other necessary attributes.
  - Hitpoint = 40
  - baseAttack = 5
- "attack" method: Calculates and returns the damage dealt by the monster's attack(total damage = baseAttack).
- "attacked" method: Takes the damage dealt by the player and subtracts it from the monster's hit points.
- "checkStatus" method: Displays the current hit points and other relevant information for the monster.
- "dead" method: Displays a message indicating that the monster has been defeated.

Use a while loop to continue the battle until one of the combatants is defeated.

Use appropriate control flow statements, such as if-else or switch statements, to handle the player's and the monster's actions during their respective turns.

Utilize the provided Scanner object to accept user and monster input for the player's actions.

Example Output:
```
Welcome to Monster Battle Warrior II
Input Player's Name: Atra
Input Monster's Name: Dimas

Atra's Status:
Level: 1
HP: 100
Attack: 10

Dimas's Status:
HP: 40
Attack: 5

Player's Turn
1
Ouch! Dimas hit for 10 damage!

Monster's Turn
1
Ouch! Atra hit for 5 damage!

...
```

```
Monster's Turn
1
Ouch! Atra hit for 5 damage!

Player's Turn
1
Ouch! Dimas hit for 10 damage!

Dimas Defeated!
Atra reached level 2
```